Kentico

MVC 6 introduction

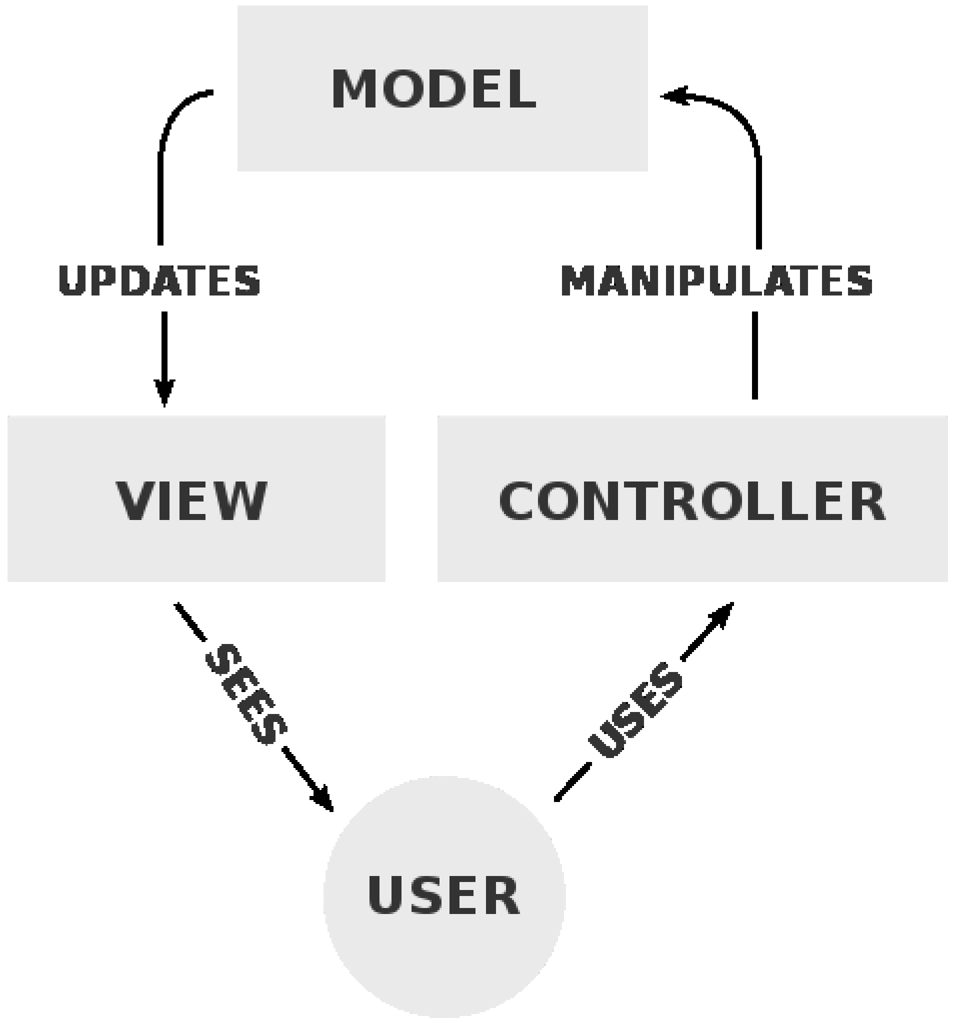
Slavomír Moroz
2016

MVC Pattern

- **M**odel-**V**iew-**C**ontroller architectural pattern for user-interface design (1970 – 1980)
- Ruby on Rails, ASP.NET MVC, Spring MVC, …

Model – state of an aspect of the application
View – displays user interface using provided model
Controller – handles user interaction by amending model and passing it to a view

https://docs.asp.net/en/latest/mvc/overview.html



Routing

- Convention-based routing

```
routes.MapRoute(name: "Default", template: "{controller=Home}/{action=Index}/{id?}");
```

- Attribute routing

```
[Route("api/[controller]")]
public class ProductsController : Controller
{
  [HttpGet("{id}")]
  public IActionResult GetProduct(int id)
  {
    ...
  }
}
```

https://docs.asp.net/en/latest/mvc/controllers/routing.html

Controllers

- In ASP.NET MVC, a *Controller* is used to define and group a set of actions. An *action* (or *action method*) is a method on a controller that handles incoming requests.
- New controller instance for each request
- Actions should return an instance of IActionResult (or Task<IActionResult> for async methods) that produces a response.
- https://docs.asp.net/en/latest/mvc/controllers/actions.html#defining-actions

Views

- ASP.NET Core MVC views are *.cshtml* files stored by default in a *Views* folder within the application.

- When an action returns a view, a process called *view discovery* takes place.
  - Unless a specific view file is specified, the runtime looks for a controller-specific view first, then looks for matching view name in the *Shared* folder.
    1. Views/<ControllerName>/<ViewName>.cshtml
    2. Views/Shared/<ViewName>.cshtml

  - When an action returns the View method, like so return View();, the action name is used as the view name
  - A view name can be explicitly passed to the method (return View("SomeView");).

https://docs.asp.net/en/latest/mvc/views/overview.html

Razor

```
@if (true) {
    WriteLiteral("<p>Test</p>");
}

@if (true) {
    <p>Text</p>
}

@if (true) {
    @:This is text.
}

@if (true)
{
    <Text>This is also text.</Text>
}
```

```
@if (condition1) {
    if (condition2) { <p>Text</p> }
}


@if (condition1) {
    <div>
        @if (condition2) { @:Text }
    </div>
}
```

https://docs.asp.net/en/latest/mvc/views/razor.html

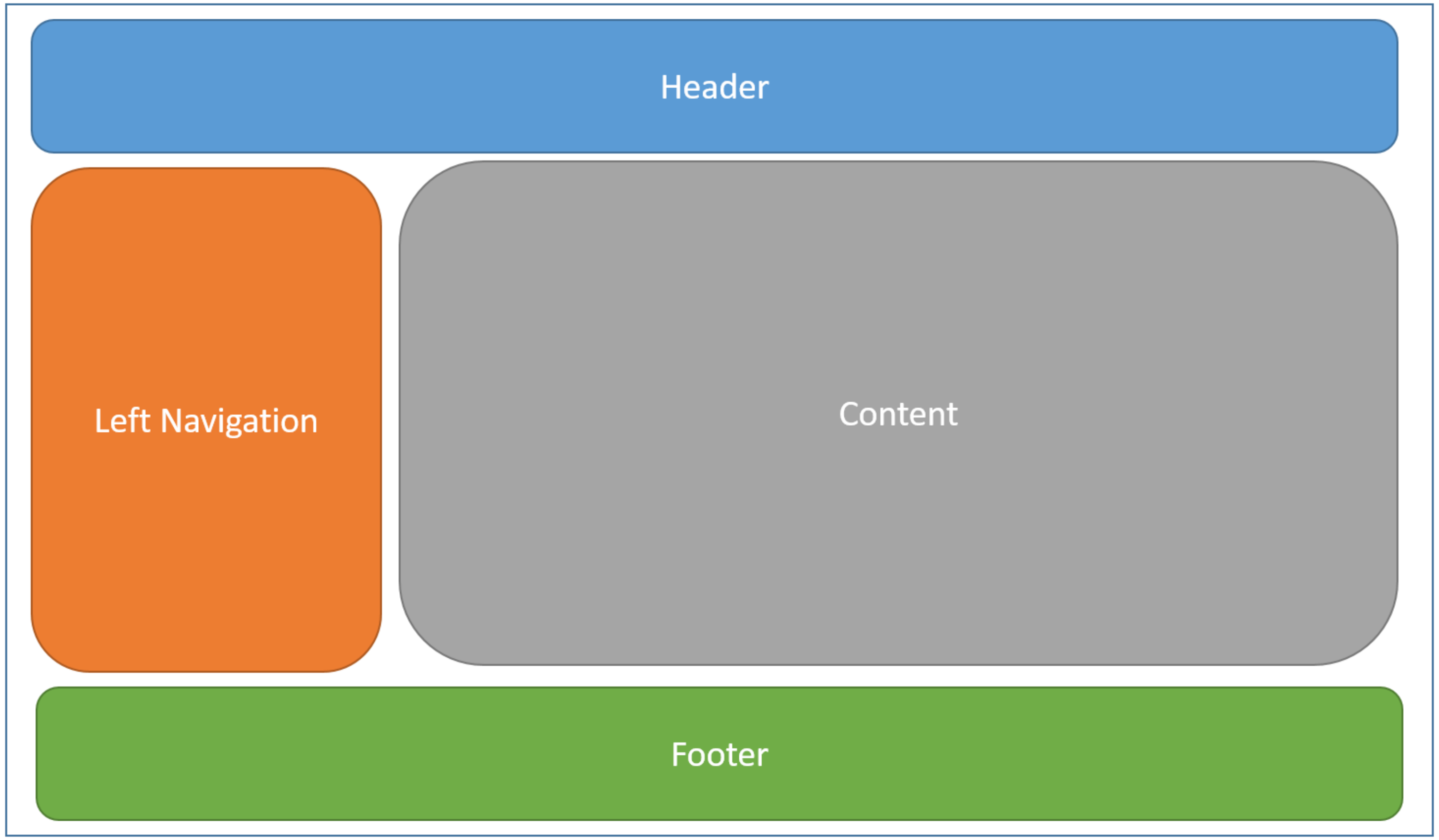
ViewData (ViewDataDictionary Class)

- Represents a container that is used to pass data between a controller and a view
- Controllers writes the data, view reads.

- ViewData.Model – passed model
- ViewData.ModelMetadata – set o information about model
- ViewData.ModelState – validation messages
- ViewData[“something”] – additional data
    - also accessible via ViewBag.

https://msdn.microsoft.com/en-us/library/system.web.mvc.viewdatadictionary(v=vs.118).aspx

Layouts



Layouts

- Specifying a Layout

```
@{
    Layout = "_Layout";
}
```

- Layout must call RenderBody. Wherever the call to RenderBody is placed, the contents of the view will be rendered.

- Running Code Before Each View

If you have code you need to run before every view, this should be placed in the _ViewStart.cshtml file.

https://docs.asp.net/en/latest/mvc/views/layout.html

Sections

- *RenderSection(…)* in layout (usually)
- Used for view-specific HTML that is not part of body (*RenderBody()*)
    - Side bars, adds, action-specific content
    - Scripts (~/Views/Shared/_Layout.cshtml)
- Not required to be defined in all views (usually, *required: false*)

View model

- *Models* folder
- Represent state of a particuallar aspect of the application
- Each controller should have a sub-folder for its models
    - Models are usually named by corresponding Actions
- Always prefer strongly-typed model to a *ViewBag* or *ViewData*
- *Models should be POCO objects without any business logic*

https://en.wikipedia.org/wiki/Plain_Old_CLR_Object

View helpers

```
@model MyProject.Models.Product

@using (Html.BeginForm())
{
    <div>
        @Html.LabelFor(m => p.Name, "Name:")
        @Html.TextBoxFor(m => p.Name)
    </div>
    <input type="submit" value="Create" />
}
```

```
@model MyProject.Models.Product
@addtaghelper "Microsoft.AspNet.Mvc.TagHelpers"

<form asp-controller="Products" asp-action="Create"
method="post">
    <div>
        <label asp-for="Name">Name:</label>
        <input asp-for="Name" />
    </div>

    <input type="submit" value="Save" />
</form>
```

https://docs.asp.net/en/latest/mvc/views/tag-helpers/index.html
https://msdn.microsoft.com/en-us/library/system.web.mvc.htmlhelper_methods(v=vs.118).aspx

Route constraints

- Route constraints generally inspect the route value associated via the route template and make a simple yes/no decision about whether or not the value is acceptable.

- Avoid using constraints for **input validation**, because doing so means that invalid input will result in a 404 (Not Found) instead of a 400 with an appropriate error message.

- template: “edit/{id:int}“
- defaults: new { controller=“IdObjectController", action=“Edit" }

- template: “edit/{guid:guid}“
- defaults: new { controller=“GuidObjectController", action=“Edit" }

- https://docs.asp.net/en/latest/fundamentals/routing.html#route-constraint-reference

Model validation (server)

Setup

- Data annotations validation attributes
  - Required, DisplayName, StringLength, Range…
  - Custom attribute that inherits ValidationAttribute.
- Model implementing IValidatableObject
- Custom : ViewData.ModelState.AddModelError()

Check

```
if (!ViewData.ModelState.IsValid)
{
    return View(model);
}

repository.Save();
return RedirectToAction("Detail", new { id = id });
```

https://docs.asp.net/en/latest/mvc/models/validation.html

Model validation (client)

- Unobtrusive validation (linked with JQuery)
- Supports only attribute validators (doesn't support IValidatableObject)
    - Custom attributes that implements IClientModelValidator
- Hard to localize

Setup

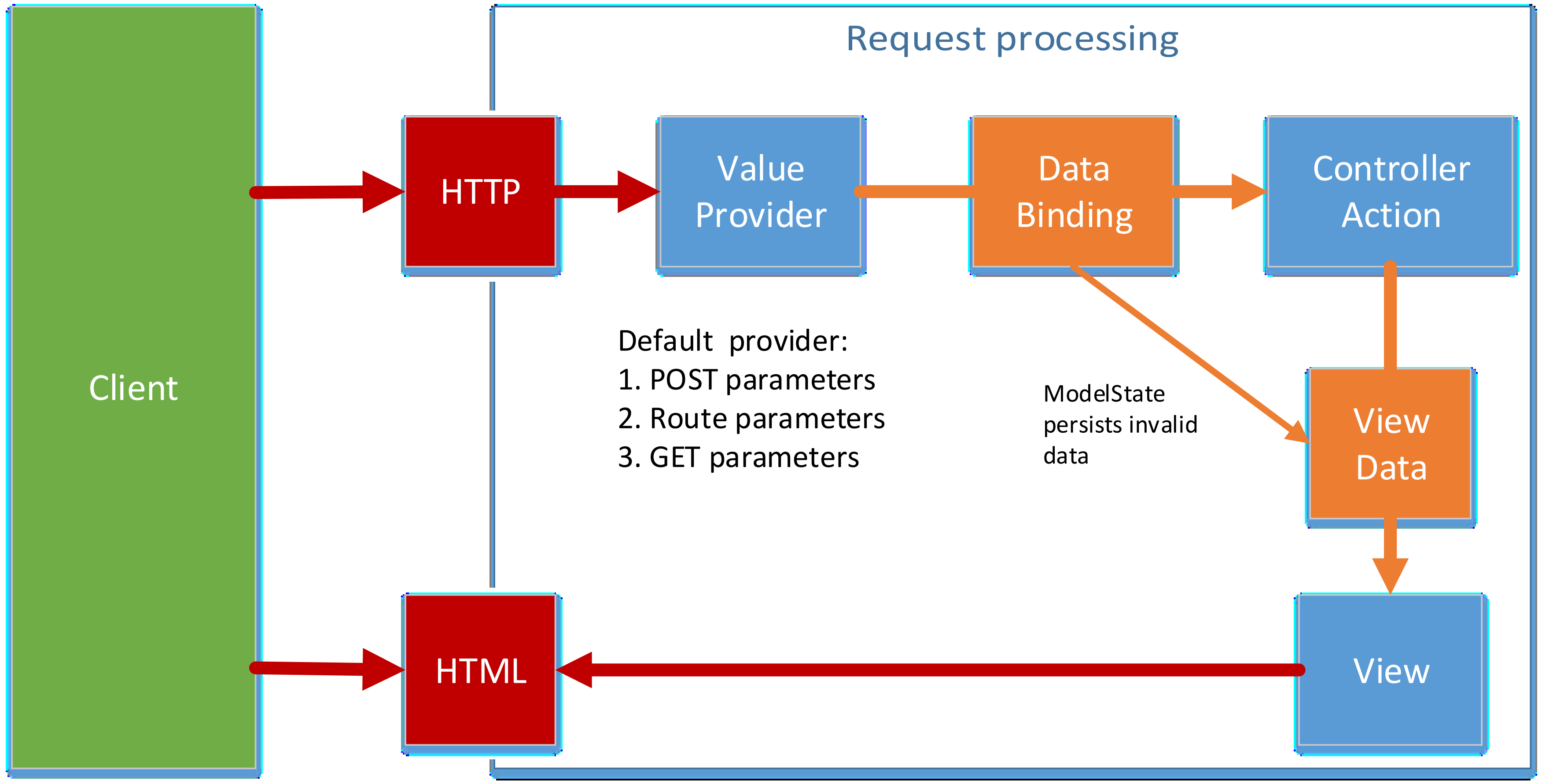
- Install nugget package Microsoft.JQuery.Unobtrusive.Validation
- Link scripts in your layout:
    - JQuery
    - JQuery-validate
    - JQuery-validate-unobtrusive

Partial views

- *Html.Partial(…)* – displays view of given name (no controller/action is called)
- *Html.RenderAction(…)* – calls a controller's action, rendering its result
    - always use *PartialViewResult*
- Usually view name starts with an underscore

https://docs.asp.net/en/latest/mvc/views/partial.html

Model binding - passing data



Customize binding behavior

- MVC contains several attributes that you can use to direct its default model binding behavior to a different source.
  - [BindRequired]: This attribute adds a model state error if binding cannot occur.
  - [BindNever]: Tells the model binder to never bind to this parameter.
  - [FromHeader], [FromQuery], [FromRoute], [FromForm]: Use these to specify the exact binding source you want to apply.
  - [FromServices]: This attribute uses dependency injection to bind parameters from services.
  - [FromBody]: Use the configured formatters to bind data from the request body. The formatter is selected based on content type of the request.
  - [ModelBinder]: Used to override the default model binder, binding source and name.

https://docs.asp.net/en/latest/mvc/models/model-binding.html

Collections binding

Primitive type array

```
ActionResult Edit(string[] array) {…}
```

PostData

```
array="John"
array="Mark"
array="Zoey"
```

Index array (complex type)

```
ActionResult Edit(Employee[] array) {…}
```

PostData

```
array[0].FirstName="John"
array[0].LastName="Smith"
array[1].FirstName="Zoey"
array[1].LastName="Castillo"
```

Dictionary

```
ActionResult Edit(
    Dictionary<string, Employee> empls
)
```

PostData

```
employees[Emp1035].FirstName="John"
employees[Emp1035].LastName="Smith"
employees[Emp2535].FirstName="Zoey"
employees[Emp2535].LastName="Castillo"
```

AntiForgeryToken

- Protection against CSRF attacks.
- Render token in form
  - generates the anti-forgery token with the Form Tag Helper
  - <form asp-action="Edit">
- Validation in controller
  - [ValidateAntiForgeryToken] attribute

https://en.wikipedia.org/wiki/Cross-site_request_forgery

Context

- HttpContext (IHttpContextAccessor service dependency injection)
    - Request
    - Response
    - Session data
    - Authentized user

- ActionContext (IActionContextAccessor service dependency injection)
    - Action metadata
    - ModelState
    - RouteData
        - Encapsulates information about a route.
        - Route parameters and values

Resources

https://docs.asp.net/en/latest/mvc/overview.html

- Complete documentation
- MVC part is well written
- Still in progress, some parts are missing

https://docs.asp.net/en/latest/tutorials/first-mvc-app/controller-methods-views.html

- Quick start tutorial